GraphQL

Expliqué par

David Gilson

https://github.com/gilsdav

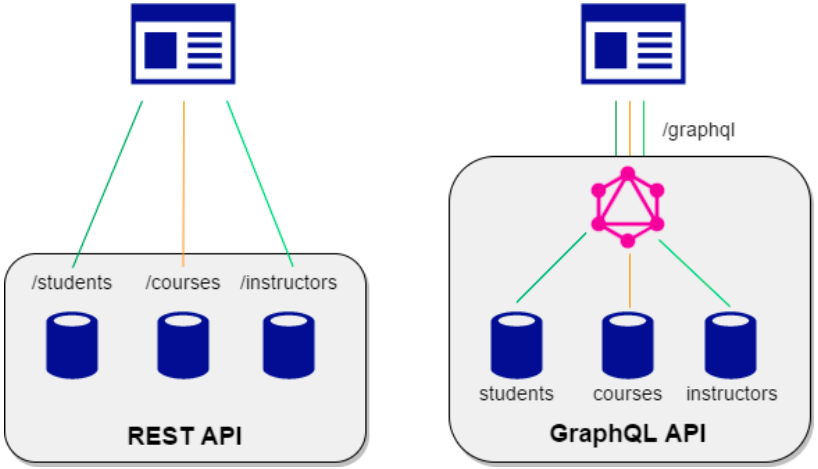
Est-ce répandu ?



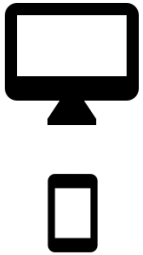


GraphQL

- A query language for your API
- Fini de faire plusieurs APIs pour la même ressource
- Front-end agnostic



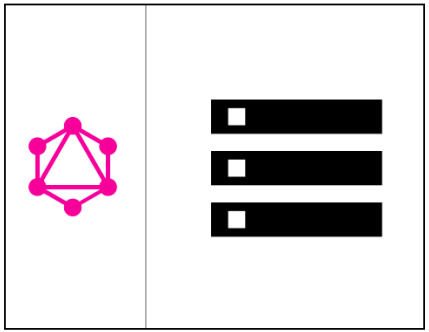

1
HTTP GET
{
"user": {
"id": "er3tg439frjw"
"name": "Mary",
"address": { … },
"birthday": "July 26, 1982"
}
}
/users/<id>
/users/<id>/posts
/users/<id>/followers
2
HTTP GET
{
"posts": [{
"id": "ncwon3ce89hs"
"title": "Learn GraphQL today",
"content": "Lorem ipsum … ",
"comments": […],
}]
}
/users/<id>
/users/<id>/posts
/users/<id>/followers
3
{
"followers": [{
"id": "leo83h2dojsu"
"name": "John",
"address": { … },
"birthday": "July 26, 1982"
},
…]
}
HTTP GET
/users/<id>
/users/<id>/posts
/users/<id>/followers



```
query {
  User(id: “er3tg439frjw”) {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```

HTTP POST

```
{
  “data”: {
    “User”: {
      “name”: “Mary”,
      “posts”: [
        { title: “Learn GraphQL today” }
      ],
      “followers”: [
        { name: “John” },
        { name: “Alice” },
        { name: “Sarah” },
      ]
    }
  }
}
```



Protocols

- HTTP
- WebSocket

- Aucun problème de compatibilité

Glossaire

- **Query**
  - Récupération (HTTP - POST)
- **Mutation**
  - Création (HTTP - POST)
  - Modification (HTTP - POST)
- **Subscription**
  - Écoute d'un évènement (WebSocket)
  - Récupération de données en temps réel (WebSocket)

GraphQL en .NET Core

- GraphQL.Server
- https://github.com/graphql-dotnet/server

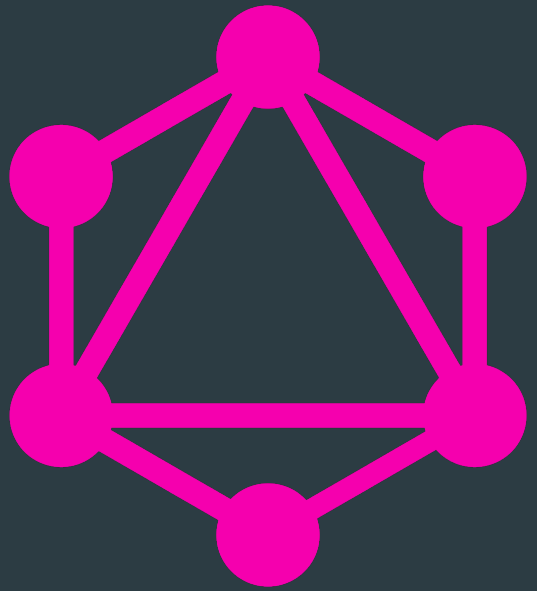
- Modules
  - GraphQL.Server.Core
  - GraphQL.Server.Transports.AspNetCore => HTTP
  - GraphQL.Server.Transports.AspNetCore.SystemTextJson => Json deserializer
    - Peut aussi être NewtonsoftJson
  - GraphQL.Server.Transports.WebSockets => WebSocket
  - GraphQL.Server.Authorization.AspNetCore => Sécurité

Stabilité

- Version preview pour .NET Core 3
  - Breaking changes mais pas fondamentaux
- Pas de problème de stabilité système rencontré

Cas d’étude

- API de création de pizza
  - https://github.com/gilsdav/dotnet-graphql-workshop
- Fournir les pizzas
- Créer/mettre à jour une pizza
- Émettre un évènement à la création/modification d’une pizza
- Protéger l’API par une session
- Stocker les données via EntityFramework



Types

Implémentation - types

- DTO/Mapping de sortie obligatoire (serveur vers client)
- Étendre la classe ObjectGraphType<MyEntity>
  - MyEntity = type source
- Appeler la fonction Field(…) pour chaque champ à créer dans le constructeur de la classe

```
Field(expression, nullable)
```

- Expression: « déstructurer » un objet du type donné au ObjectGraphType.

```
x => x.Id
```

Implémentation - types

- Nom de l'attribut = nom de l'attribut source
  - Sauf si utilisation de la méthode `.Name("OtherName")`
- Possibilité de mettre une description

```
.Description("Description de l'attribut")
```

- Et encore d'autres possibilités

```
.DefaultValue("default").DeprecationReason("Because of default value")
```

Implémentation - types

```
public class ToppingType : ObjectGraphType<Topping>
{
    public ToppingType() {
        Field(x => x.Id).Description("ToppingId");;
        Field(x => x.Name, false).Description("Name of topping");
    }
}
```

```
public class Topping
{
    [Key]
    public int Id { get; set; }
    [Required(ErrorMessage = "Name is required")]
    public string Name { get; set; }
    public ICollection<PizzaTopping> PizzaToppings { get; set; }
}
```

Implémentation - types

- Field plus complexe (avec manipulation)

```
Field(name, resolver)
```

- Le resolver doit retourner un autre type (un autre DTO/Mapping)

```
public class PizzaType : ObjectGraphType<Pizza>
{
    public PizzaType() {
    Field(x => x.Id);
    Field(x => x.Name, false).Description("Name of pizza");
    Field<ListGraphType<ToppingType>>("toppings", resolve: context =>
        context.Source.PizzaToppings.Select(pt => pt.Topping).ToList());
    }
}
```

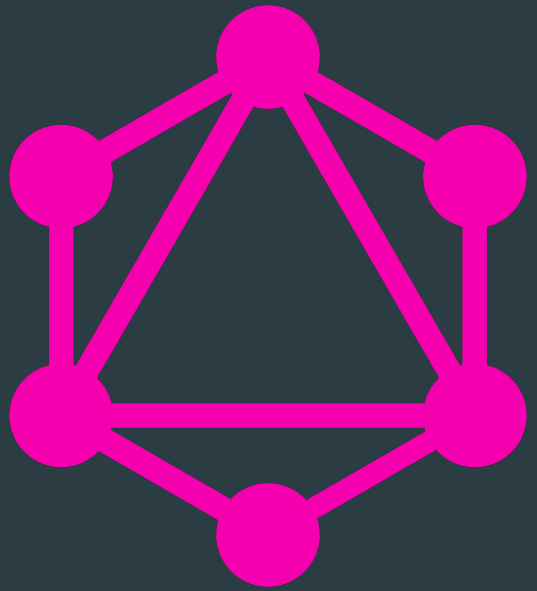
Implémentation – inputTypes

- DTO/Mapping d'entrée obligatoire (client vers serveur)
- Étendre la classe InputObjectGraphType
- Appeler la fonction Field(…) pour chaque champ à créer dans le constructeur de la classe

```
Field<Type>(name)
```

Implémentation – inputTypes

```
public class PizzaInputType : InputObjectGraphType
{
    public PizzaInputType()
    {
        Name = "PizzaInputType";
        Field<IntGraphType>("id");
        Field<NonNullGraphType<StringGraphType>>("name");
        Field<NonNullGraphType<ListGraphType<IntGraphType>>>("toppings");
    }
}
```



Query

Query

- Étendre la classe ObjectGraphType
  - Du déjà vu non ?
- Appeler la fonction Field(…) pour chacune des query dans le constructeur de la classe

```
Field<OutputType>(
    "queryName",
    arguments,
    resolve
);
```

- Arguments: paramètre de la query pouvant être fourni par le client

Query

```
public PizzaQuery(IPizzaRepository pizzaRepository) {
    Field<PizzaType>(
        "pizza",
        arguments: new QueryArguments(
            new QueryArgument<NonNullGraphType<IntGraphType>> { Name = "id" }
        ),
        resolve: context => this.loadPizza(context, pizzaRepository)
    );
}
```

Query dynamique

- Récupération des données en arguments
  - Le contexte contient un dictionnaire « Arguments »

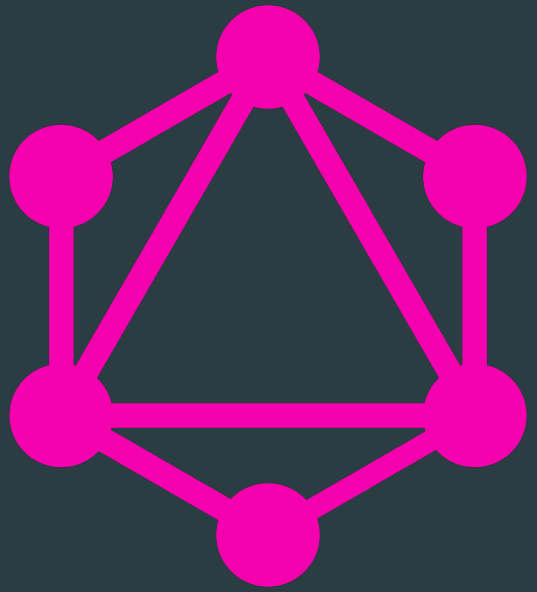
```
context.Arguments["argumentName"]
```

- Conditionner la query selon les fields demandés
  - Le contexte contient un dictionnaire « SubFields »

```
context.SubFields.FirstOrDefault(kv => kv.Key == "fieldName").Key != null
```

Query dynamique

```
private Pizza loadPizza(
    IResolveFieldContext<object> context,
    IPizzaRepository pizzaRepository)
{
    int id = (int)context.Arguments["id"];
    var loadToppings = context.SubFields
        .FirstOrDefault(kv => kv.Key == "toppings").Key != null;
    return pizzaRepository.GetById(id, loadToppings);
}
```



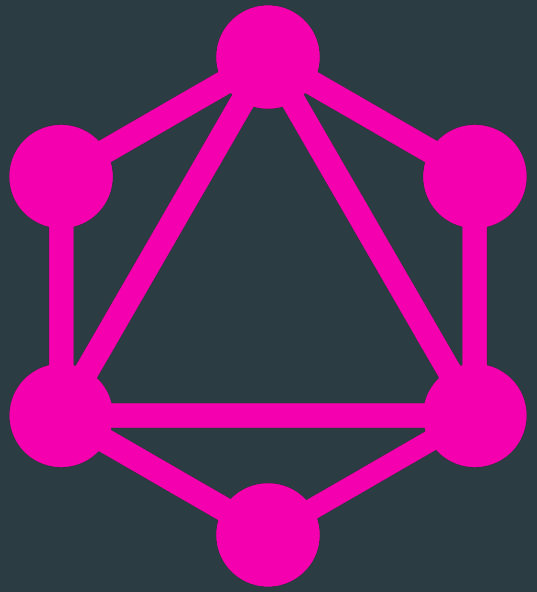
Mutation

Mutation

- Étendre la classe ObjectGraphType
- Appeler la fonction Field(…) pour chacune des mutations dans le constructeur de la classe

```
Field<OutputType>(
    "mutationName",
    arguments,
    resolve
);
```

- *Oui, c'est exactement comme pour les queries*



Subscription

Subscription

- Étendre la classe ObjectGraphType
- Appeler la fonction AddField (…) pour chacune des souscriptions dans le constructeur de la classe

```
AddField(new EventStreamFieldType
{
    Name = "subscriptionName",
    Type = typeof(OutputType),
    Resolver = new FuncFieldResolver<MyEntity>(resolverFn),
    Subscriber = new EventStreamResolver<MyEntity>(subscribeFn)
});
```

Subscription

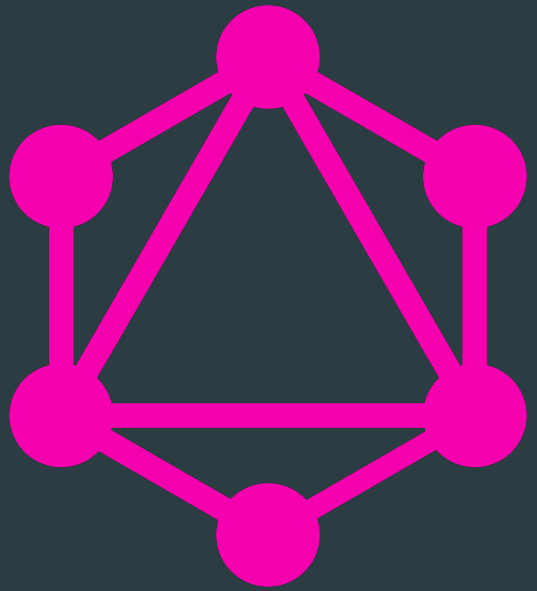
- Resolver: permet de résoudre une donnée par rapport à l'évènement émis
  - Donnée directe
  - Récupération de données en DB
  - …
- Subscriber: observable à écouter
  - RX (Reactive API) ça vous parle ? On parle bien de RXJS pour .NET Core

Subscription

```
public MySubscription(IEventsService eventsService)
{
    _eventsService = eventsService;
    AddField(new EventStreamFieldType
    {
        Name = "subscriptionName",
        Type = typeof(OutputType),
        Resolver = new FuncFieldResolver<MyEntity>(Resolve),
        Subscriber = new EventStreamResolver<MyEntity>(Subscribe)
    });
}

private MyEntity Resolve(IResolveFieldContext context)
{
     return context.Source as MyEntity;
}

private IObservable<MyEntity> Subscribe(IResolveEventStreamContext context)
{
    return _eventsService.ListenMyEntityEvent();
}
```



Schema

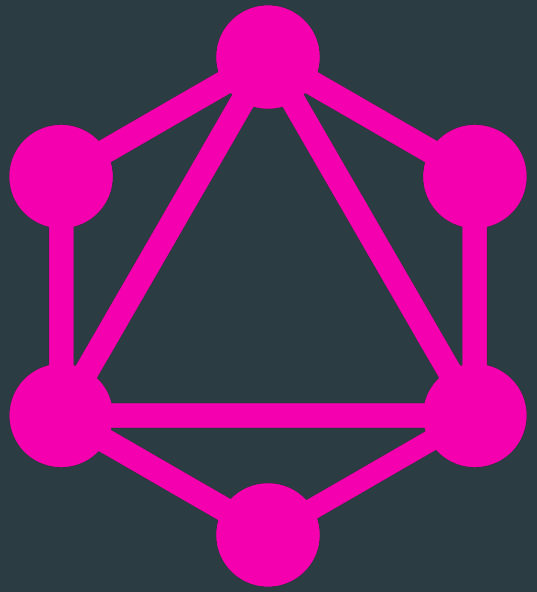
Schema

- Étendre la classe Schema
- Initialiser les attributs:
  - Query
  - Mutation
  - Subscription
- L’attribut « Query » est le seul requis

Schema

```
public class PizzaSchema: Schema
{
    public PizzaSchema(
        PizzaQuery query,
        PizzaMutation mutation,
        PizzaSubscription subscription)
    {
        Query = query;
        Mutation = mutation;
        Subscription = subscription;
    }
}
```

- J'utilise la DI mais vous pouvez également créer l'instance de chaque ici



Configuration

Configuration - services

```
services.AddGraphQL()
    .AddSystemTextJson(
        deserializerSettings => { },
        serializerSettings => { })
    .AddWebSockets() // For subscriptions
    .AddGraphTypes(ServiceLifetime.Scoped)
});
```

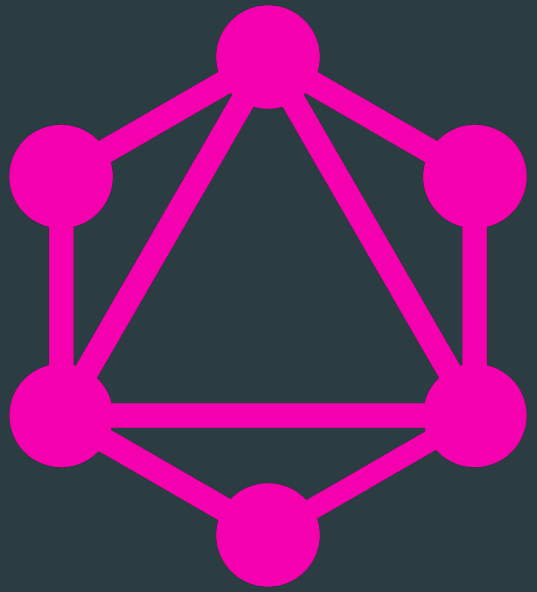
Configuration - middlewares

```
app.UseWebSockets();
app.UseGraphQLWebSockets<PizzaSchema>("/graphql");
app.UseGraphQL<PizzaSchema>("/graphql");
```

Configuration - UI

- Pour tester facilement: GraphQLPlayground
- Compatible avec l'authentification: GraphiQL

```
if (env.IsDevelopment())
{
    app.UseGraphQLPlayground(new GraphQLPlaygroundOptions());
    app.UseGraphiQLServer(new GraphiQLOptions
    {
        Path = "/ui/graphiql/pizza",
        GraphQLEndPoint = "/graphql"
    });
    app.UseGraphiQLServer(new GraphiQLOptions
    {
        Path = "/ui/graphiql/security",
        GraphQLEndPoint = "/auth"
    });
}
```



Sécurité

Sécurité - services

- Ajouter cette configuration sur services.AddGraphQL()

```
.AddGraphQLAuthorization(options => {
    options.AddPolicy("LoggedIn", p =>
        p.RequireAuthenticatedUser());
    options.AddPolicy("Bob", p =>
        p.RequireClaim(ClaimTypes.Name, "Bob"));
});
```

Sécurité – ressource sécurisation

- Ajouter cette configuration sur Field()

```
.AuthorizeWith("LoggedIn");
```

- Les ressources n’ayant pas cette configuration sont publiques

Références

- https://github.com/graphql-dotnet/server
- https://graphql-dotnet.github.io/docs/getting-started/introduction
- https://graphql.org/
- https://github.com/gilsdav/dotnet-graphql-workshop